

P2C:

Understanding Output Data Files via On-the-Fly Transformation from Producer to Consumer Execution

Yonghwi Kwon, Fei Peng, Dohyeong Kim, Kyungtae Kim, Xiangyu Zhang, Dongyan Xu

Department of Computer Science, Purdue University

Vinod Yegneswaran

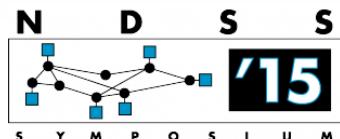
SRI International

John Qian

Cisco Systems



SRI International



Motivation

- *Understanding unknown data files and network messages is a prominent security challenge*



Downloads and Installs
few freeware programs

Weeks later



Found mysterious
binary files

Motivation

- *Without audit logging/monitoring systems*
 - Who created the files?
 - Do they contain private data?
 - Personal profile
 - Contact list
 - Key-strokes



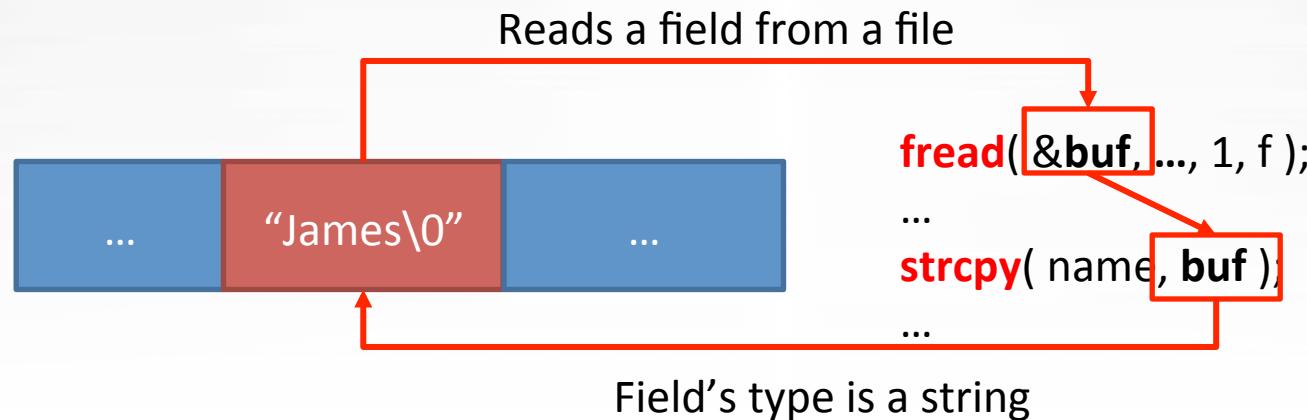
Found mysterious
binary files

00000130 BA 0B F5 26 AC F7 0F F5 A6 AC 0F 08 F5 66 AC 9F 1B EA 27
00000143 B0 3E 38 D4 9B B3 9E EB AF 16 AC E7 FA EB 44 D6 73 FD 75
00000156 12 FB B9 FE 6A C9 7A AE BF 5A B1 DE 2D D4 5B B3 DE 23 D4
00000169 B3 D6 7B C1 1A 29 AC 0E F5 36 AC F7 09 F5 B6 AC 9F 11
0000017E EA 65 1E 97 87 7A 3B D6 7A FD D5 9E F5 5C 7F 9D CA 7A AE
0000018FBF 3C B0 9E 0D 3D 0C F5 CE AC B7 0F F5 2E AC 77
000001A2 08 F1 AE A7 77 0C F5 6E AC 77 0E 75 DE 2E
000001B5 20 BA B6 7A 4C 06 73 F0 38 B8 5C FF F4 66 3D D7 3F 7D
000001C8 1F C9 ED 17 7B 07 3B BE A7 67 D0 06 34 E6 E6 EB 6E
000001DB 89 7C F5 8D 15 6F B4 7C F9 CA 61 BF B0 9C 46 C1 5F 49 39
000001EE 11 B1 25 C1 01 56 5A 4E 07 C6 E9 91 9D 02 FF BE EC 5F 1A
00000201 1A F7 0D FC B9 3F A2 02 7E 8C B1 F8 FD E8 40 3A 6F E2 6D
00000214 14 40 C9 59 8C B1 78 67 85 FE 04 40 C9 D9 6C 1E 05 09 94
00000227 90 4D F9 9F 3C 22 AC DF 05 FE 01 02 A5 2E F0 0F EB EF CF
0000023A E6 F1 04 94 F6 27 FF D1 B9 F9 E7 04 FE 61 FC 73 02 FF 30
0000024D FE 01 C2 3A B9 FA 07 70 FE 08 86 F1 07 06 FE 61 FC 81 81
00000260 7F 18 5C 5C 24 47 72 FB F5 20 42 BE 3D C8 F8 83 B8 BE 89
00000273 57 05 FE 01 70 FC D6 36 84 F5 DC 79 36 94 F5 DC 79 36
00000286 8C F5 DC 7D E5 3C D6 73 FB E9 F9 AC E7 CE EB 0B 58 CF 9D
00000299 D7 C3 59 CF 9D A7 23 58 CF 9D 17 23 59 CF 9D 17 A3 58 CF
000002AC 9D 17 A3 59 CF 9D 77 63 58 CF 9D 77 E3 09 EB E4 FA 7F 3C

The first two lines of the hex dump are circled in black.

Motivation

- *Existing solutions require consumer programs*
 - Input format reverse engineering based on consumers
 - Prospex, Tupni, REWARDS, Dispatcher, Reformat, ...
 - Monitoring the execution of consumer to analyze how the files/messages are parsed



Motivation

- *What if we only have the producer program?*
 - Botnet Command and Control (C&C) protocol
 - No consumer is present on the victim machine
 - Consumer exists on the attacker's server



Motivation

- *What if we only have the producer program?*
 - Botnet Command and Control (C&C) protocol
 - No consumer is present on the victim machine
 - Consumer exists on the attacker's server



Observation

- Producer and consumer are **symmetric**
 - “**Checking Conformance of a Producer and a Consumer**”, Evan Driscoll, Amanda Burton, and Thomas Reps, FSE’11
 - The **correctness of a producer** can be verified by checking its conformance to the **corresponding consumer**

Observation

- Producer and consumer are **symmetric**
 - Consumer and Producer follow the same rule
 - ((boolean, double, boolean) | (boolean))*

Producer

```
1 sendReading(Sensor* device, int prev)
2 if device->setting == prev then
3   writeBool(false);
4 else
5   writeBool(true);
6   writeDouble(device->setting);
7   writeBool(device->valid);
```

Consumer

```
1 updateReading(int* setting, bool* valid)
2   *setting = readDouble();
3   *valid = readBool();
4 main()
5   int setting;
6   bool valid;
7   while ... do
8     if readBool() then
9       updateReading (&setting, &valid);
10      ... // do something with current readings
```

Observation

- Producer and consumer are **symmetric**
 - Consumer and Producer follow the same rule
 - ((boolean, double, boolean) | (boolean))*

Producer

```
1 sendReadings();
2 if device == "A" {
3   writeBoolean(true);
4 } else {
5   writeBoolean(false);
6   writeDouble(1.23);
7   writeBoolean(true);
```

Consumer

```
valid)
```

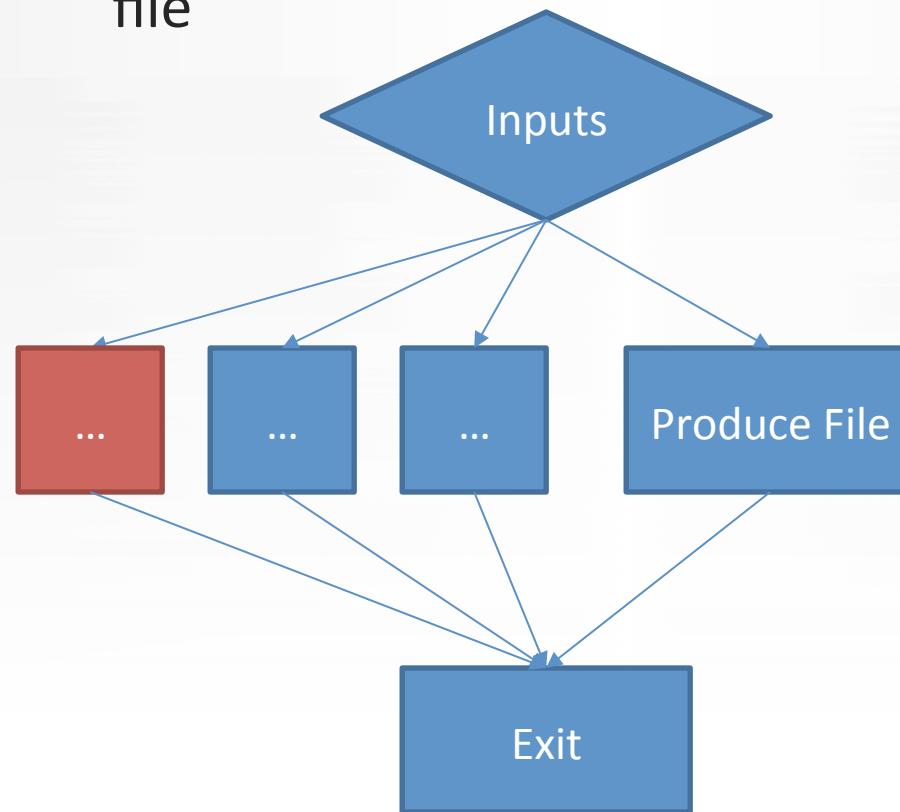
Our idea:

Run producer to *create* consumer (P2C)

```
8 if readBool() then
9   updateReading(&setting, &valid);
10 ... // do something with current readings
```

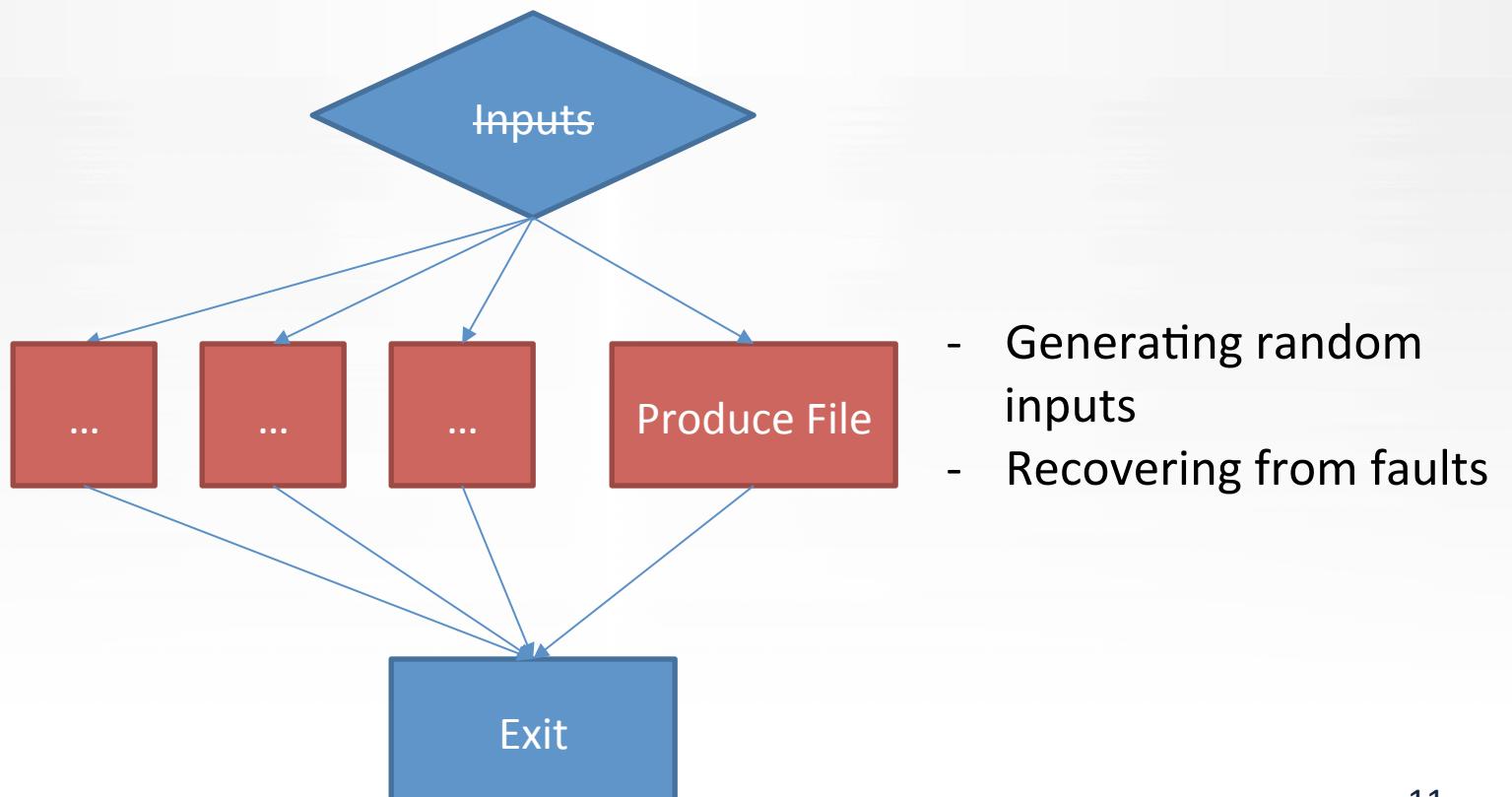
Forced execution

- Executing producer w/o proper inputs
 - May do different tasks and quit, without creating a file



Forced execution

- Our earlier tool comes to the rescue
 - Our forced execution engine: X-force



Overview

- Basic idea
 - Given an **unknown file/message** and a **potential producer**



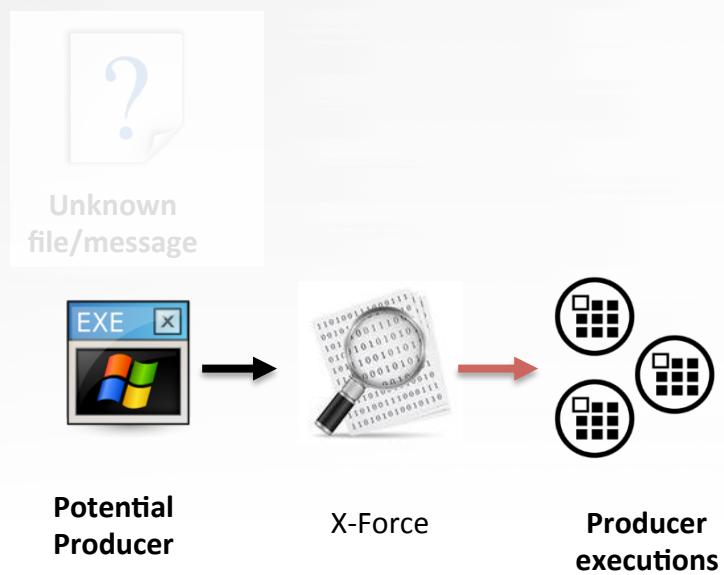
Unknown
file/message



Potential
Producer

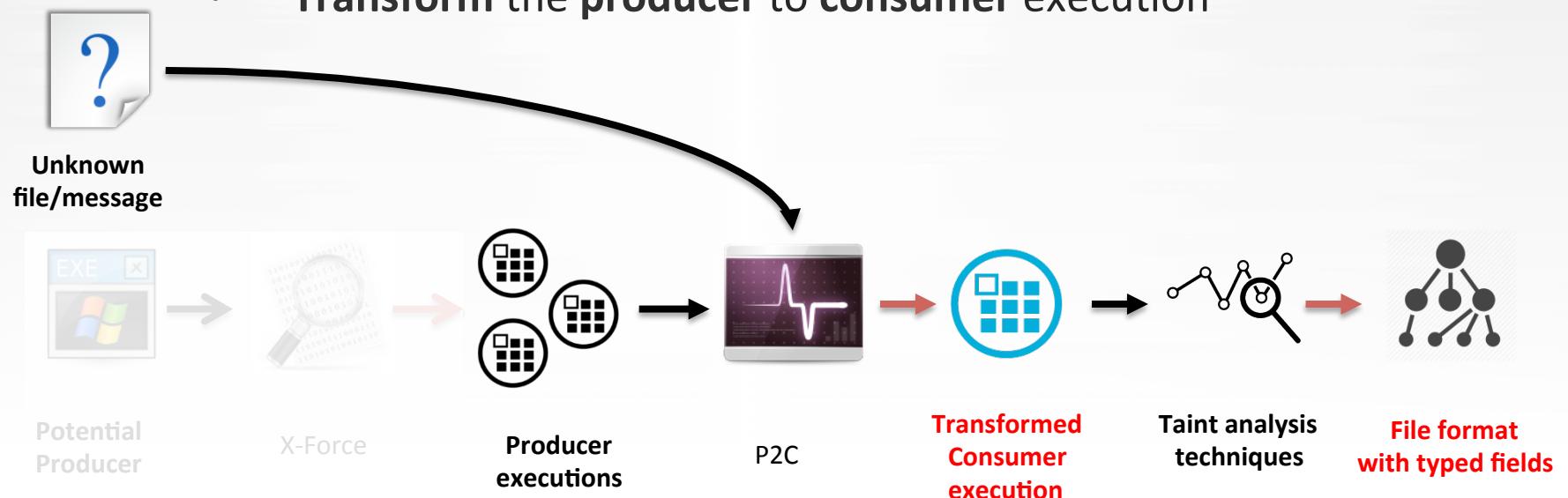
Overview

- Basic idea
 - Given an **unknown file/message** and a **potential producer**
 - Explore execution paths that contain “file open-for-write” operations (Producer executions)
 - Leveraging a binary forced execution technique (X-force)



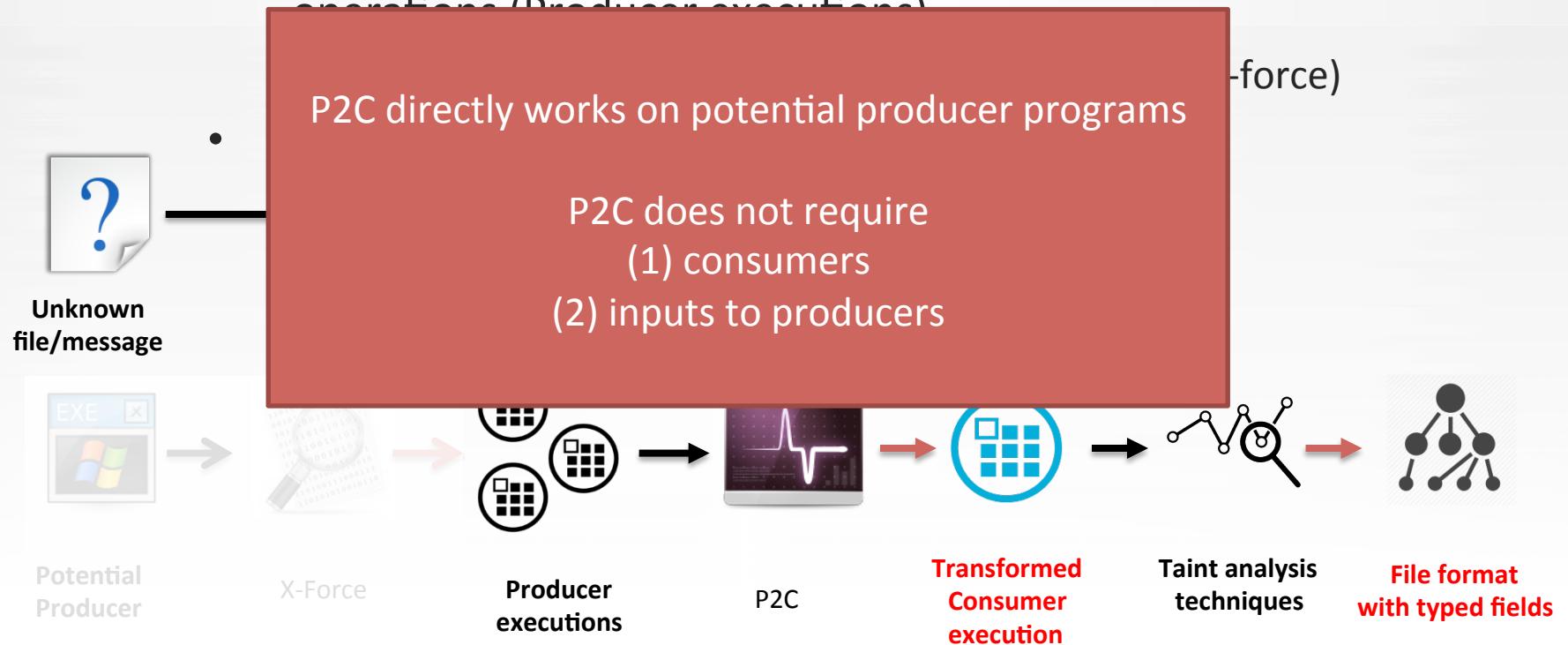
Overview

- Basic idea
 - Given an **unknown file/message** and a **potential producer**
 - Explore execution paths that contain “file open-for-write” operations (Producer executions)
 - Leveraging a binary forced execution technique (X-force)
 - **Transform the producer to consumer execution**



Overview

- Basic idea
 - Given an **unknown file/message** and a **potential producer**
 - Explore execution paths that contain “file open-for-write” operations (Producer executions)



An unknown binary file

0x4bfd

0x5

0x15

0x2

0x2

“silver_member”

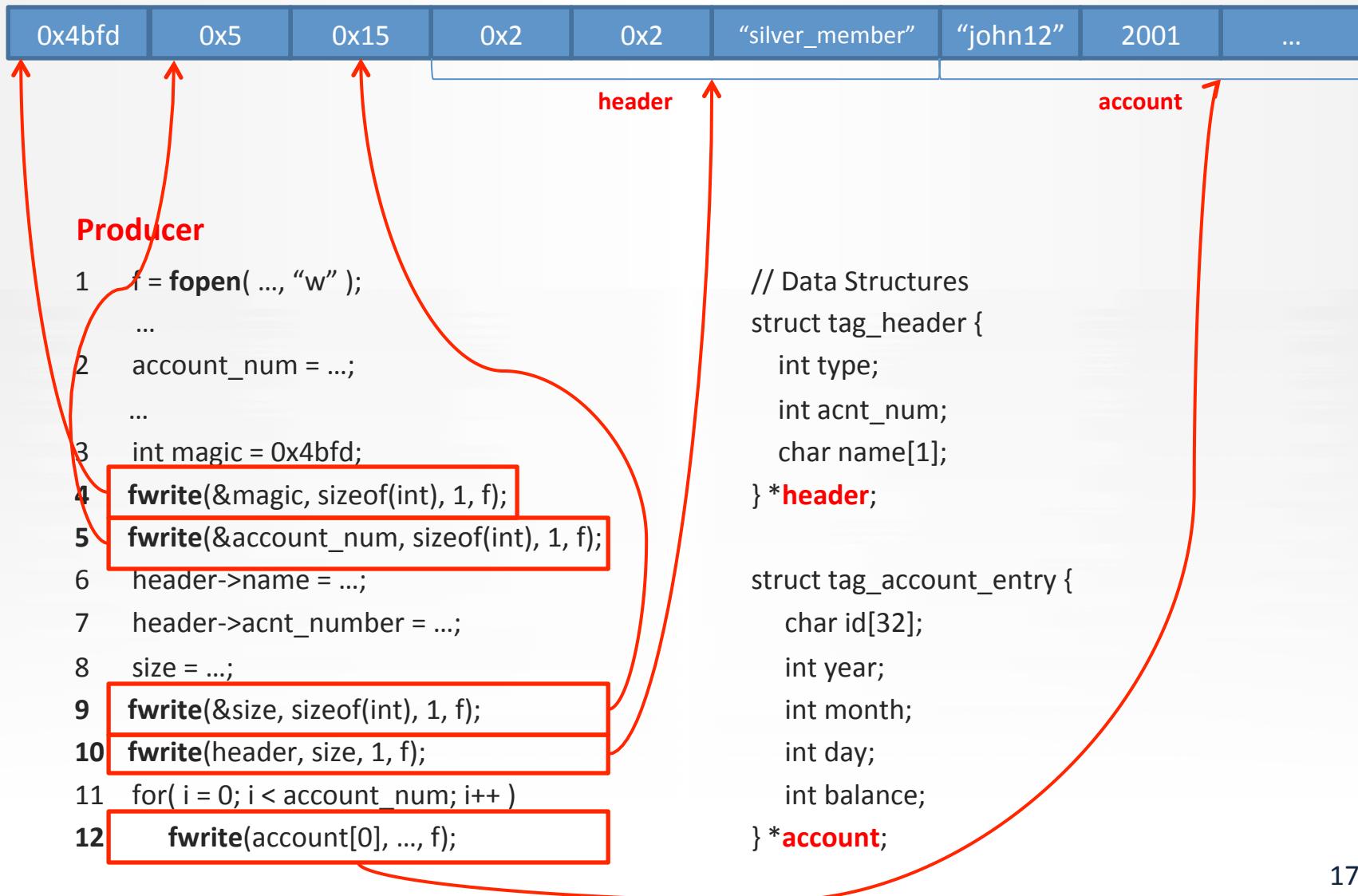
“john12”

2001

...

How it is generated?

- Unknown file



Producer w/o proper inputs (X-force)

- Unknown file



Producer

```
1   f = fopen( ..., "w"
2   ...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "# "
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fwrite(account[0], ..., f);
```

Randomly generated

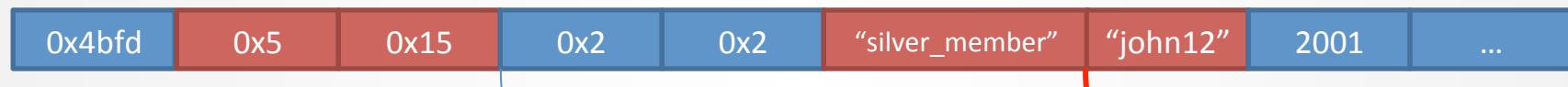
Randomly generated

```
// Data Structures
struct tag_header {
    int type;
    int acnt_num;
    char name[1];
} *header;

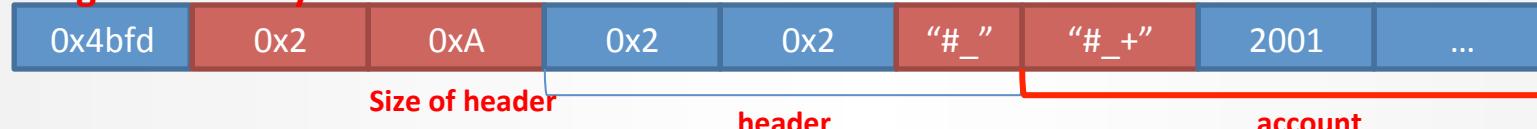
struct tag_account_entry {
    char id[32];
    int year;
    int month;
    int day;
    int balance;
} *account;
```

Producer w/o proper inputs (X-force)

- Unknown file



- File generated by the forced execution



Producer

```
1  f = fopen( ..., "w" );
...
2  account_num = ...; // 2
...
3  int magic = 0x4bfd;
4  fwrite(&magic, sizeof(int), 1, f);
5  fwrite(&account_num, sizeof(int), 1, f);
6  header->name = ...; // "#_"
7  header->acnt_number = ...; // 2
8  size = ...;
9  fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12     fwrite(account[0], ..., f);
```

```
// Data Structures
struct tag_header {
    int type;
    int acnt_num;
    char name[1];
} *header;

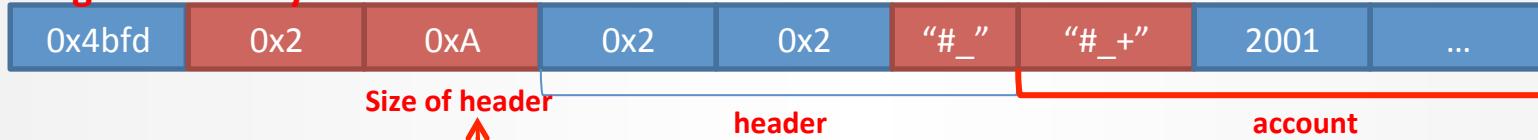
struct tag_account_entry {
    char id[32];
    int year;
    int month;
    int day;
    int balance;
} *account;
```

Producer w/o proper inputs (X-force)

- Unknown file



- File generated by the forced execution



Producer

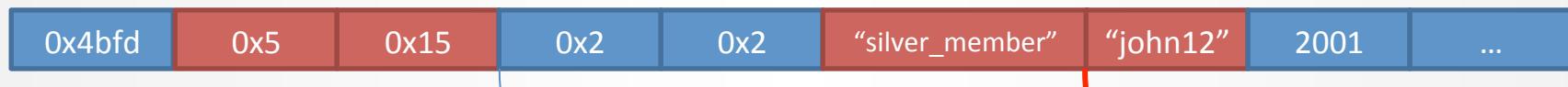
```
1  f = fopen( ..., "w" );
...
2  account_num = ...; // 2
...
3  int magic = 0x4bfd;
4  fwrite(&magic, sizeof(int), 1, f);
5  fwrite(&account_num, sizeof(int), 1, f);
6  header->name = ...; // "#_"
7  header->acnt_number = ...; // 2
8  size = ...;
9  fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12     fwrite(account[0], ..., f);
```

```
// Data Structures
struct tag_header {
    int type;
    int acnt_num;
    char name[1];
} *header;

struct tag_account_entry {
    char id[32];
    int year;
    int month;
    int day;
    int balance;
} *account;
```

Producer w/o proper inputs (X-force)

- Unknown file



- File generated by the forced execution



Producer

```
1  f = fopen( ..., "w" );
...
2  account_num = ...; // 2
...
3  int magic = 0x4bfd;
4  fwrite(&magic, sizeof(int), 1, f);
5  fwrite(&account_num, sizeof(int), 1, f);
6  header->name = ...; // "#_"
7  header->acnt_number = ...; // 2
8  size = ...;
9  fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12     fwrite(account[0], ..., f);
```

```
// Data Structures
struct tag_header {
    int type;
    int acnt_num;
    char name[1];
} *header;

struct tag_account_entry {
    char id[32];
    int year;
    int month;
    int day;
    int balance;
} *account;
```

Transforming a producer to a consumer

- Unknown file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

- File generated by the forced execution

0x4bfd	0x2	0xA	0x2	0x2	“#_”	“#_+”	2001	...
--------	-----	-----	-----	-----	------	-------	------	-----

Producer

```
1 f = fopen( ..., "w" );
...
2 account_num = ...; // 2
...
3 int magic = 0x4bfd;
4 fwrite(&magic, sizeof(int), 1, f);
5 fwrite(&account_num, sizeof(int), 1, f);
6 header->name = ...; // "#_"
7 header->acnt_number = ...; // 2
8 size = ...;
9 fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12     fwrite(account[0], ..., f);
```

Consumer

```
1 f = fopen( ..., "r" );
...
2 account_num = ...; // 2
...
3 int magic = 0x4bfd;
4 fread(&magic, sizeof(int), 1, f);
5 fread(&account_num, sizeof(int), 1, f);
6 header->name = ...; // "#_"
7 header->acnt_number = ...; // 2
8 size = ...;
9 fread(&size, sizeof(int), 1, f);
10 fread(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12     fread(account[0], ..., f);
```



Transforming a producer to a consumer

- Unknown file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

- File generated by the forced execution

0x4bfd	0x2	0xA	0x2	0x2	“#_”	“#_+”	2001	...
--------	-----	-----	-----	-----	------	-------	------	-----

Producer

```
1   f = fopen( ..., "w" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fwrite(account[0], ..., f);
```

Consumer

```
1   f = fopen( ..., "r" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fread(account[0], ..., f);
```



Transforming a producer to a consumer

- Unknown file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

- File generated by the forced execution

0x4bfd	0x5	0xA	0x2	0x2	“#_”	“#_+”	2001	...
--------	-----	-----	-----	-----	------	-------	------	-----

Producer

```
1   f = fopen( ..., "w" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // “#_”
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fwrite(account[0], ..., f);
```

Consumer

```
1   f = fopen( ..., "r" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);
6   header->name = ...; // “#_”
7   header->acnt_number = ...; // 2
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fread(account[0], ..., f);
```



Transforming a producer to a consumer

- Unknown file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

- File generated by the forced execution

0x4bfd	0x5	0x15	0x2	0x2	“#_”	“#_+”	2001	...
--------	-----	------	-----	-----	------	-------	------	-----

Producer

```
1   f = fopen( ..., "w" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fwrite(account[0], ..., f);
```

Consumer

```
1   f = fopen( ..., "r" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fread(account[0], ..., f);
```



Transforming a producer to a consumer

- Unknown file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

- File generated by the forced execution

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“#_+”	2001	...
--------	-----	------	-----	-----	-----------------	-------	------	-----

Producer

```
1   f = fopen( ..., "w" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fwrite(account[0], ..., f);
```

Consumer

```
1   f = fopen( ..., "r" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fread(account[0], ..., f);
```



Transforming a producer to a consumer

- Unknown file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

- File generated by the forced execution

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	...
--------	-----	------	-----	-----	-----------------	----------	------	-----

Producer

```
1   f = fopen( ..., "w" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12      fwrite(account[0], ..., f);
```

Consumer

```
1   f = fopen( ..., "r" );
...
2   account_num = ...; // 2
...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);   
6   header->name = ...; // "#_"
7   header->acnt_number = ...; // 2
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )   
12      fread(account[0], ..., f);   
```



Technical Challenges

- *Unexposed field correlations*
 - *Symmetric read operations do not help*
 - *Semantically related read operations do not have explicit dependence*

Producer

```
1 f = fopen( ... );  
2 ...  
3 len = strlen(buf);  
4 ...  
5 fwrite(&len, sizeof(int), 1, f);  
6 fwrite(buf, strlen(buf), 1, f);
```



Consumer

```
1 f = fopen( ... );  
2 ...  
3 len = strlen(buf);  
4 ...  
5 fread(&len, sizeof(int), 1, f); // (1)  
6 fread(buf, strlen(buf), 1, f); // (2)  
7 // no dependence path between (1) and (2)
```

Technical Challenges

- *Unexposed field correlations*
 - *Original Definition Tracking*
 - *Change the original definition of variables*
 - *Patching and Tracing*
 - *Iteratively run the program for consistency checking*
 - *Inconsistency indicates the presence of unexposed field correlations*
 - *Try a different value (e.g. try to read more bytes)*

Evaluation

- 5 programs generate files / 4 programs generate network messages*

Program	Program Size	Unknown file size	# of iteration (Matched / Unmatched)	# of typed fields/ # of fields	Elapsed Time	Coverage / # of total inst.	# of paths explored
InfoZip	37KB	5KB	12 / 0	19 / 35	2m 45s	6015/6015	385
Steganography	17KB	4219KB	24 / 0	3 / 3	9m 27s	631/974	12
FreePiano	2296KB	6KB	12 / 0	6 / 6	28m 35s	47991/286571	2531
Mp3gain	109KB	1304KB	17 / 0	6 / 10	24m 42s	21672/21672	754
Yamdi	225KB	102KB	38 / 0	34 / 72	1h 16m 38s	21491/24577	1183
Zbot	26KB	30Bytes	7 / 2	5 / 7	8m 48s	3089/3089	199
Winping	244KB	45Bytes	5 / 0	4 / 8	2m 27s	14758/29620	983
PowerOn	25KB	102Bytes	17 / 0	2 / 2	22s	709/1058	29
NetworkMorris	1049KB	52Bytes	20 / 4	15 / 39	8m 6s	3057/7814	186

Evaluation

- *Matched: When the file is correctly parsed*
- *Unmatched: When it fails to parse the file*

Program	Program Size	Unknown file size	# of iteration (Matched / Unmatched)	# of typed fields/ # of fields	Elapsed Time	Coverage / # of total inst.	# of paths explored
InfoZip	37KB	5KB	12 / 0	19 / 35	2m 45s	6015/6015	385
Steganography	17KB	4219KB	24 / 0	3 / 3	9m 27s	631/974	12
FreePiano	2296KB	6KB	42 / 0	6 / 6	28m 35s	47991/286571	2531
Mp3gain	109KB	1304KB	17 / 0	6 / 10	24m 42s	21672/21672	754
Yamdi	225KB	102KB	38 / 0	34 / 72	1h 16m 38s	21491/24577	1183
Zbot	26KB	30Bytes	7 / 2	5 / 7	8m 48s	3089/3089	199
Winping	244KB	45Bytes	5 / 0	4 / 8	2m 27s	14758/29620	983
PowerOn	25KB	102Bytes	1 / 0	2 / 2	21s	709/1058	29
NetworkMorris	1049KB	52Bytes	20 / 4	15 / 39	8m 6s	3057/7814	186

Evaluation

- *# of typed fields / total # of fields*

Program	Program Size	Unknown file size	# of iteration (Matched / Unmatched)	# of typed fields/ # of fields	Elapsed Time	Coverage / # of total inst.	# of paths explored
InfoZip	37KB	5KB	12 / 0	19 / 35	2m 45s	6015/6015	385
Steganography	17KB	4219KB	24 / 0	3 / 3	9m 27s	631/974	12
FreePiano	2296KB	6KB	42 / 0	6 / 6	28m 35s	47991/286571	2531
Mp3gain	109KB	1304KB	17 / 0	6 / 10	24m 42s	21672/21672	754
Yamdi	225KB	102KB	38 / 0	34 / 72	1h 16m 38s	21491/24577	1183
Zbot	26KB	30Bytes	7 / 2	5 / 7	8m 48s	3089/3089	199
Wining	244KB	45Bytes	5 / 0	4 / 8	2m 27s	14758/29620	983
PowerOn	25KB	102Bytes	1 / 0	2 / 2	21s	709/1058	29
NetworkMorris	1049KB	52Bytes	20 / 4	15 / 39	8m 6s	3057/7814	186

Evaluation

- *Elapsed time(P2C alone): 21s ~ 1h 16m*

Program	Program Size	Unknown file size	# of iteration (Matched / Unmatched)	# of typed fields/ # of fields	Elapsed Time	Coverage / # of total inst.	# of paths explored
InfoZip	37KB	5KB	12 / 0	19 / 35	2m 45s	6015/6015	385
Steganography	17KB	4219KB	24 / 0	3 / 3	9m 27s	631/974	12
FreePiano	2296KB	6KB	42 / 0	6 / 6	28m 35s	47991/286571	2531
Mp3gain	109KB	1304KB	17 / 0	6 / 10	24m 42s	21672/21672	754
Yamdi	225KB	102KB	38 / 0	34 / 72	1h 16m 38s	21491/24577	1183
Zbot	26KB	30Bytes	7 / 2	5 / 7	8m 48s	3089/3089	199
Winping	244KB	45Bytes	5 / 0	4 / 8	2m 27s	14758/29620	983
PowerOn	25KB	102Bytes	1 / 0	2 / 2	21s	709/1058	29
NetworkMorris	1049KB	52Bytes	20 / 4	15 / 39	8m 6s	3057/7814	186

Case study

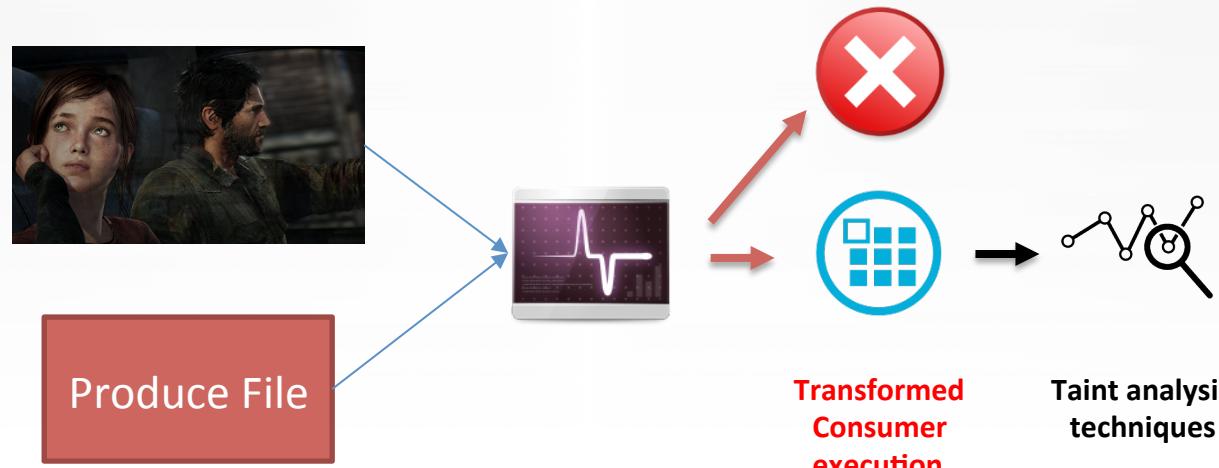
- *Steganography*
 - Embeds a secret text by changing the LSB(Least Significant Bit)s



Is there a hidden message or not?

Evaluation

- *Steganography*
 - Apply P2C to transform producer to consumer
 - Use existing taint analysis techniques to understand the secret message format



Case study

- Steganography
- ...

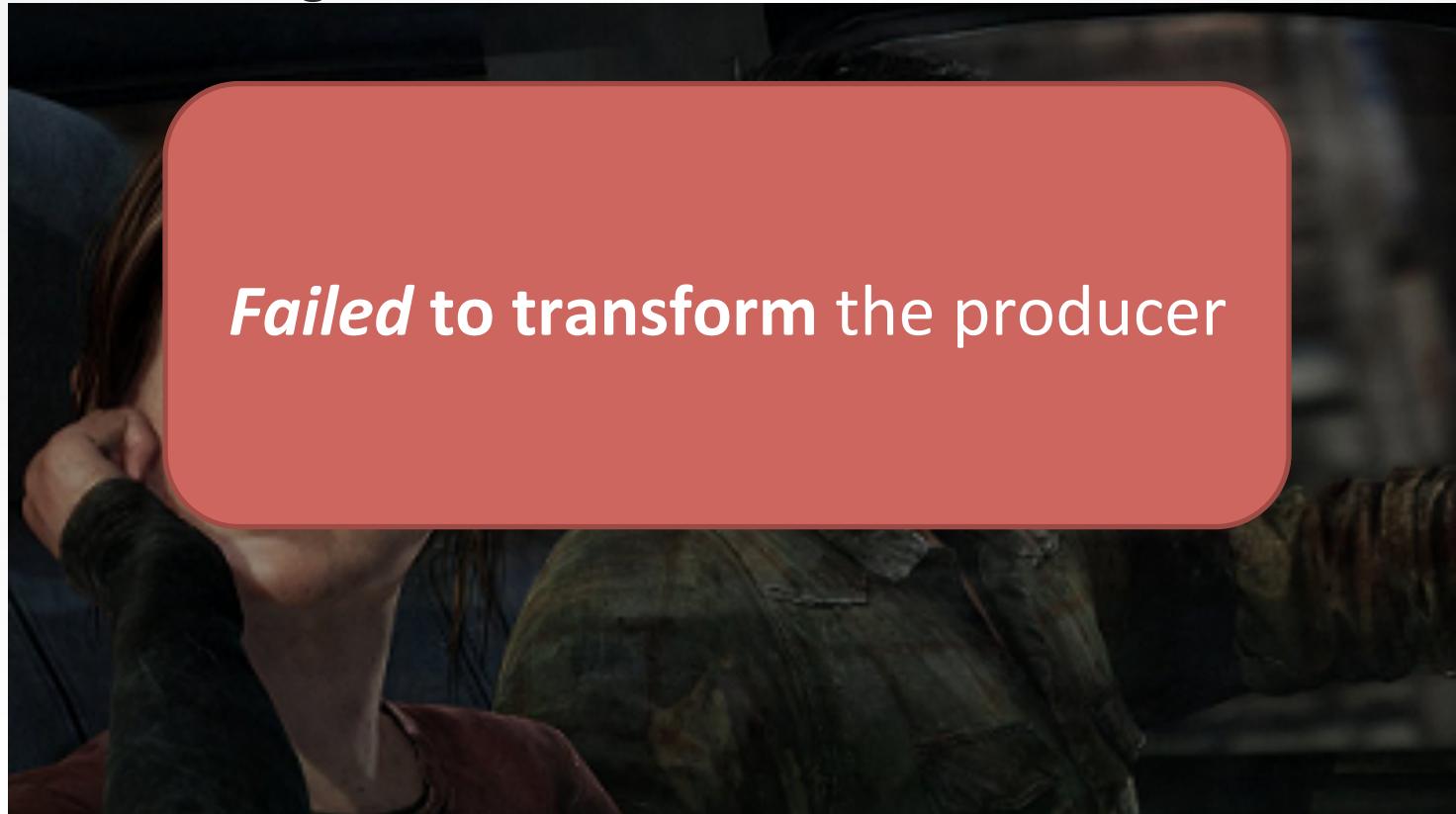
“LAST OF NDSS”



Is there a hidden message or not?

Case study

- *Steganography*
 - Benign



Is there a hidden message or not?

Evaluation

- *Observations*
 - *# of iterations to find the correct transformation is not large*
 - *In most cases, program dependences are exposed.*
 - *Can precisely identify all the fields in the given files/message*
 - *Yamdi: Floating Point*
 - *NetworkMorris/WinPing/InfoZip: fields are not related to any standard API*
 - *Transformed consumer execution recognizes more fields than some typical consumers (mp3tag)*

Conclusion

- *P2C: an output format reverse-engineering tool*
 - *Understand the structure and meaning of unknown file/message*
 - *Based only on producer, without consumer*
- *Key Idea*
 - *Transforming a producer execution to the ideal consumer execution*
- *Key Features*
 - *Highly accurate*
 - *No need to know exact producer*
 - *No need to know how to run it*

Question?

Thank you!

Email: yongkwon@purdue.edu